VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Implementace semihladké Newtonovy metody

On implementation of the semi-smooth Newton method

2016 Lukáš Kresta

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Zadání bakalářské práce

Student: **Lukáš Kresta**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 1103R031 Výpočetní matematika

Téma: Implementace semihladké Newtonovy metody
On implementation of the semi-smooth Newton method

Jazyk vypracování: čeština

Zásady pro vypracování:

Řada aplikací vede na řešení soustavy rovnic s nelineárními diferencovatelnými funkcemi. Pro takovou úlohu můžeme použít Newtonovu metodu. V případě, že řešíme soustavu rovnic s nelineárními funkcemi, které nejsou diferencovatelné, musíme použít jinou metodu. Takovouto metodou je semihladká Newtonova metoda a je založena na využití Clarkeova kalkulu.
Práce se zaměřuje na pochopení této metody, její implementaci a použití této metody ve vhodných aplikacích.

Práce bude mít tyto části:
Soustava rovnic s nelineárními nediferencovatelnými rovnicemi
Clarkeův kalkul
Semihladká Newtonova metoda
Implementace semihladké Newtonovy metody
Testování algoritmu na vhodných úlohách

Seznam doporučené odborné literatury:

M. Hintermüller: Semismooth Newton Methods and Applications, Oberwolfach-Seminar on "Mathematics of PDE-Constrained Optimization" at Mathematisches Forschungsinstitut in Oberwolfach, November 2010

Dále dle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Beremlijski, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 29.04.2016

doc. RNDr. Jiří Bouchala, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2016 ..... Vrecha .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 1. dubna 2016 .....

Rád bych na tomto místě poděkoval Ing. Petru Beremlijskému, Ph.D. , za jeho pomoc při tvorbě, a všem autorům mých literárních zdrojů.

Abstrakt

Cílem mé bakalářské práce je implementace semihladké Newtonovy metody. Newtonova metoda je numerickou metodou, která se využívá při řešení soustavy rovnic. Tato metoda si ovšem neporadí s řešením rovnic, které obsahují nediferencovatelné funkce. Právě z tohoto důvodu si zavedeme semihladkou Newtonovu metodu, která je založena na využití Clarkeova kalkulu.
V této práci postupně rozebírám Clarkeův kalkul, pod který spadá zobecněná derivace se zobecněným gradientem a zobecněný Jacobián. Dále popisuju "klasickou" Newtonovu metodu a hlavně semihladkou Newtonovu metodu. Poté popisuju samotnou implementaci semihladké Newtonovy metody v programu Matlab, dále srovnávám řešení Newtonovou metodou pomocí analyticky spočteného zobecněného Jacobiánu a numericky spočteného zobecněného Jacobiánu. Na závěr řeším praktickou úlohu, v které zkoumám deformaci struny při kontaktu s překážkou.

Klíčová slova: Newtonova metoda, Clarkeův kalkul, deformace struny, zobecněná derivace, zobecněný Jacobián, zobecněný gradient, lipschitzovská funkce

Abstract

The goal of this work is an implementation of semismooth Newton method. Newton method is a numerical method, which is proposed to solve the system of equations. This method is not able to solve the system of equations with nodifferentable functions. Because of it we will define semismooth Newton metod which is based on Clarke calculus.
I need some basic definitions from Clarke calculus like generalized derivative, generalized gradient and generalized Jacobian. Then I describe "classical" Newton method and mainly semismooth Newton method. The next part of my work is devoted to implementation of semismooth Newthon method in program Matlab and comparison of solutions based on generalized Jacobian and numerical approximation of generalized Jacobian. In the end of my work I show you some numerical experiments with deformation of string in contact with rigid obstacle.

Key Words: Newton method, Clarke calculus, deformation of string, generalized derivative, generalized Jacobian, generalized gradient, Lipschitz function

Obsah

Seznam použitých zkratek a symbolů

$JF(x)$	–	Jacobián funkce F v bodě x
$\partial F(x)$	–	zobecněný Jacobián funkce F v bodě x
o	–	nulový vektor
O	–	nulová matice
$\mathbb{B}$	–	jednotková koule
$f^0(x;h)$	–	zobecněná derivace funkce f v bodě x ve směru h
$\partial f(x)$	–	zobecněný gradient funkce f v bodě x
conv	–	konvexní obal
U(x,r)	–	okolí bodu x o poloměru r

Seznam obrázků

1 Úvod

Cílem mé bakalářské práce je implementace semihladké Newtonovy metody. Newtonova metoda je numerickou metodou, která se využívá při řešení soustavy rovnic. Tato metoda si ovšem neporadí s řešením rovnic, které obsahují nediferencovatelné funkce. Právě z tohoto důvodu si zavedeme semihladkou Newtonovu metodu, která je založena na využití Clarkeova kalkulu.
V kapitole Clarkeův kalkul popíšu, co je lipschitzovská spojitost. Zadefinuju zobecněnou derivaci a popíšu vlastnosti zobecněné derivace lipschitzovsky spojité funkce. Zadefinuju také zobecněný gradient, zobecněný Jacobián a pojmy, které nám mohou pomoci vypočítat zobecněný gradient, či zobecněný Jacobián. Ukážu také pár příkladů na výpočet zobecněného gradientu a zobecněného Jacobiánu. Poté následuje kapitola Newtonova metoda, ve které nejprve popíšu základní věci u "klasické"Newtonovy metody a poté popíšu semihladkou Newtonovu metodu. V této kapitole mám také definici semihladkosti, podmínky lokální a globální konvergence semihladké Newtonovy metody. V další kapitole, kterou je Implementace semihladké Newtonovy metody, popisuju implementaci semihladké Newtonovy metody v programu Matlab. Nejprve popíšu implementaci výpočtu zobecněného Jacobiánu a implementaci kontroly funkce. Poté popisuju, po částech, implementaci semihladké Newtonovy metody. Jako poslední kapitolu před závěrem mám kapitolu Numerické testování. Nejprve testuji rychlost výpočtů semihladké Newtonovy metody při použití analyticky spočteného Jacobiánu a při využití numericky spočteného Jacobiánu. Dále řeším praktickou úlohu, deformaci struny při kontaktu s překážkou. Nejprve si popíšu tento příklad a zavedu si pár vztahů, co budu používat. Poté popíšu implementaci semihladké Newtonovy metody pro tento případ a provedu několik numerických testů, kdy si nechám vždy vykreslit jak se struna deformovala. Na závěr vše shrnu.

2 Clarkeův kalkul

V této kapitole popisuji definici zobecněné derivace, z které vychází i definice zobecněného Jacobiánu a definice zobecněného gradientu. Tyto pojmy se využívají při řešení úloh s nediferencovatelnými funkcemi a jsou součástí Clarkeova kalkulu. K těmto definicím si také potřebuji zavést pojem lipschitzovská spojitost, kterou popisuju v následující podkapitole. Definice v této kapitole jsem čerpal z [2], v této publikaci jsou také uvedené důkazy k pojmům, které zde popisuji.

2.1 Lipschitzovská spojitost

Věta 2.1 *Nechť ω je podmnožina z $\mathbb{R}^n$, F je funkce z ω do $\mathbb{R}^m$ a λ je nezáporné reálné číslo.*

a) Říkáme, že F je lipschitzovsky spojitá s modulem λ na ω, pokud:

$$||F(x_1) - F(x_2)|| \leq \lambda ||x_1 - x_2|| \quad \forall x_1, x_2 \in \omega. \tag{1}$$

b) F je lipschitzovsky spojitá v okolí x s modulem λ, pokud existuje $\epsilon > 0$ takové, že F je lipsichtzovsky spojitá s modulem λ na $x + \epsilon\mathbb{B}$.

c) Pokud je F lipsichtzovsky spojitá v okolí každého $x \in \omega$, tak F je lokálně lipschitzovsky spojitá v ω.

2.2 Zobecněný gradient

Abychom mohli spočítat zobecněný gradient, tak si nejprve zavedeme zobecněnou derivaci:

Definice 2.1 *Řekneme, že*

$$f^0(x;h) = \limsup_{x' \to x, t \downarrow 0} \frac{f(x'+th) - f(x')}{t} \tag{2}$$

je zobecněnou derivací funkce f v bodě x ve směru h.

Pokud je f lipschitzovsky spojitá v okolí x, je lipschitzovsky spojitá v okolí všech y, které jsou blízko x.
Nyní si tedy můžeme zavést zobecněný gradient.

Definice 2.2 *Nechť je $f : \mathbb{R}^n \to \mathbb{R}$ lipschitzovsky spojitá v okolí x. Zobecněný gradient f v bodě x, značíme jej $\partial f(x)$, je množina*

$$\partial f(x) = \{\epsilon \in \mathbb{R}^n | \langle \epsilon, h \rangle \leq f^0(x;h), \forall h \in \mathbb{R}^n\}, \tag{3}$$

kde prvky $\partial f(x)$ se nazývají subgradienty f v x.

Pro spojitě diferencovatelné f, v bodě x platí $f^0(x;h) = \langle \nabla f(x), h \rangle$ a $\partial f(x) = \{\nabla f(x)\}$.
Dále si zavedeme pomocnou funkci A.

Definice 2.3 *Pro množinu $A \in \mathbb{R}$ je funkce $\delta_A : \mathbb{R} \to \mathbb{R} \cup \infty$ definována jako*

$$\delta_A(h) = sup\{\langle a, h \rangle | a \in A\} \text{ pro } h \in \mathbb{R}^n$$

nazývána pomocnou funkcí A.

Nyní můžeme napsat, že:

$$f^0(x;h) = \delta_{\partial f(x)}(h) \forall h \in \mathbb{R}^n. \tag{4}$$

Následující tvrzení popisuje vlastnosti lipschitzovsky spojité funkce.

Věta 2.2 *Nechť $f : \mathbb{R}^n \to \mathbb{R}$ je lipschitzovsky spojitá v okolí x s modulem λ. Pak:*

a) Funkce $h \to f^0(x;h)$ je konečná, pozitivně homogenní, subaditivní na $\mathbb{R}$ a

$$|f^0(x;h)| \leq \lambda ||h||.$$

b) Funkce $f^0(\cdot;\cdot)$ je shora polospojitá v (x,h) pro každé $h \in \mathbb{R}^n$ a funkce $h \to f^0(x;h)$ je lipschitzovsky spojitá s modulem λ na $\mathbb{R}^n$.

c) $\partial f(x)$ je neprázdná konvexní kompaktní množina a $\partial f(x) \subset \lambda \mathbb{B}$.

d) Pro každé $h \in \mathbb{R}^n$, platí

$$f^0(x;h) = max\{\langle \epsilon, h\rangle | \epsilon \in \partial f(x)\}.$$

e) Multifunkce $y \to \partial f(y)$ je shora polospojitá (a uzavřená) v x.

f) $\partial F(x) \subset (\partial F^1(x), \partial F^2(x), \cdots, \partial F^m(x))^T$, kde $\partial F^i(x)$ je zobecněný gradient reálné funkce F^i v x pro $i = 1, 2, \ldots, m$, kde $F(x) = (F^1(x), F^2(x), \ldots, F^m(x))$ je vektorová funkce o velikosti m.

Ještě napíšu jednu větu k zobecněné derivaci.

Věta 2.3 *Nechť $f : \mathbb{R}^n \to \mathbb{R}$ je lipschitzovsky spojitá v okolí x. Pak pro všechna $h \in \mathbb{R}^n$ platí*

$$f^0(x;h) \leq \limsup\{\langle \nabla f(x'), h\rangle | x' \to x, x' \notin \Omega_f\},$$

kde Ω_f je podmnožina D_f, kde F není diferencovatelná.

2.3 Zobecněný Jacobián

Následující definice zobecněného Jacobiánu i Věta 2.5, která popisuje alternativní konstrukci zobecněného gradientu, jsou založeny na následujícím tvrzení.

Věta 2.4 *(Rademacherova) Nechť $F : \mathbb{R}^n \to \mathbb{R}^m$ je lokálně lipschitzovsky spojitá v $\mathbb{R}^n$, pak je F diferencovatelná "skoro všude".*

Nyní zadefinuju zobecněný Jacobián.

Definice 2.4 *Nechť $F : \mathbb{R}^n \to \mathbb{R}^m$ je lipschitzovsky spojitá v okolí x. Zobecněný Jacobián z $F : \mathbb{R}^n \to \mathbb{R}^m$ v x, značíme jej $\partial F(x)$, je podmnožina z $\mathbb{R}^{m\times n}$(matice o velikosti $[m \times n]$) kterou definujeme*

$$\partial F(x) = conv\partial_B F(x), \tag{5}$$

kde

$$\partial_B F(x) := \{\lim_{i\to\infty} JF(x_i)|x_i \to x, x_i \notin \Omega_f\}, \tag{6}$$

kde Ω_f je podmnožina D_f, kde F není diferencovatelná.

Řekněme tedy, že pro $m = 1$ se zobecněný Jacobián shoduje se zobecněným gradientem. Proto je tu jistý nesoulad ve značení ∂ pro $m = 1$ a $m > 1$, protože znakem ∂ značíme jak Jacobián tak gradient. Budu používat ∂ pro označení Jacobiánu, protože se to objevuje v obvyklé literatuře. Ještě nám chybí, abychom zadefinovali konvexní obal pro množinu $M = \{x_i\}$, $i \in I$, $I = \{1, 2 \ldots, n\}$:

Definice 2.5

$$convM = \sum_{i\in I} \lambda_i x_i, \text{ kde navíc platí } \lambda_i \geq 0, \sum_{i\in I} \lambda_i = 1. \tag{7}$$

V následujícím tvrzení si ukážeme, jak vypočítat zobecněný gradient jiným způsobem.

Věta 2.5 *Nechť je $f : \mathbb{R}^n \to \mathbb{R}$ lipschitzovsky spojitá v okolí x, pak:*

$$\partial f(x) = conv\partial_B f(x), \tag{8}$$

kde

$$\partial_B f(x) := \{\lim_{i\to\infty} \nabla f(x_i)|x_i \to x, x_i \notin \Omega_f\},$$

kde Ω_f je podmnožina D_f, kde F není diferencovatelná.

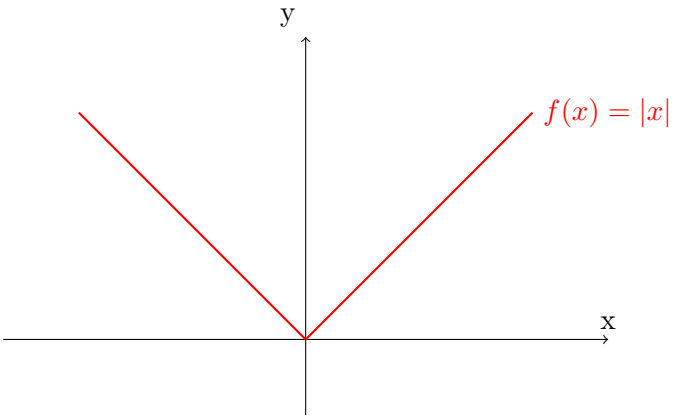
2.4 Příklady

a) Nyní ukážu, jak se dá spočítat zobecněný gradient u následujících funkcí.

i) Nejprve začneme s touto funkcí:

$$f(x) = |x|.$$

Zadaná funkce bude vypadat takto:



Funkci si rozdělíme na 3 úseky, kdy je $x > 0$, $x < 0$ a $x = 0$. K výpočtu zobecněného gradientu využijeme tohoto vztahu:

$$\partial f(x) = conv\{\lim_{i\to\infty} \nabla f(x_i) | x_i \to x, x_i \notin \Omega_f\},$$

kde Ω_f je podmnožina D_f, kde F není diferencovatelná.

a spočítáme zobecněné gradienty pro uvedené volby x. Nejprve vypočítáme zobecněný gradient, kdy je $x > 0$. Protože derivací x je 1 a limita posloupnosti 1 je 1, tak pro tuto volbu x dostáváme:

$$\partial f(x) = conv\{1\} = \{1\} = \{f'(x)\},\ x > 0.$$

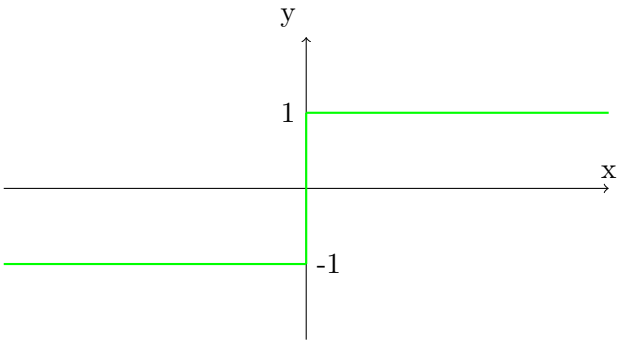
Nyní si spočítáme zobecněný gradient pro $x < 0$. V tomhle případě pracujeme s funkcí $f(x) = |x| = -x$, tudíž budeme derivovat $-x$, což je -1 a limita posloupnosti -1 je -1, tedy pro tuto část dostáváme:

$$\partial f(x) = conv\{-1\} = \{-1\} = \{f'(x)\},\ x < 0.$$

Teď nám jedině chybí spočítat zobecněný gradient, když $x = 0$. Protože v nule nemá funkce $f(x) = |x|$ derivaci, tak zkonstruujeme derivaci f, pro x jdoucí do tohoto bodu zleva a zprava. Limita zleva v $x = 0$ je -1 a limita zprava je v tomto bodě 1. Tudíž zobecněný gradient nám vyjde takto:

$$\partial f(0) = conv\{-1, 1\} = \langle -1, 1\rangle.$$

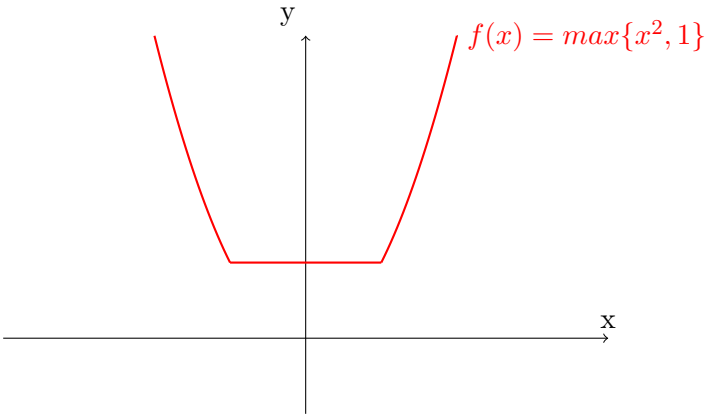
Zobecněný gradient pro funkci $f(x) = |x|$ bude vypadat následovně:



ii) Nyní spočteme zobecněný gradient pro tuto funkci:

$$f(x) = max\{x^2, 1\}$$

Tato funkce bude vypadat následovně.



U tohohle případu si funkci rozdělíme na 5 částí, na úseky kdy se $x > 1$, $x < -1$, $1 > x > -1$, $x = -1$, $x = 1$. Pro všechny bude opět platit, že jejich zobecněný gradient se rovná konvexní množině:

$$\partial f(x) = conv\{\lim_{i \to \infty} \nabla f(x_i) | x_i \to x, x_i \notin \Omega_f\},$$

kde Ω_f je podmnožina D_f, kde F není diferencovatelná.

Budeme postupovat obdobně jako u příkladu uvedeného výše. Tudíž, teď si spočítáme zobecněný gradient pro $x > 1$. Při téhlé volbě x derivace funkce x^2 je $2x$, tak je nám jasné, že:

$$\partial f(x) = conv\{2x\} = \{2x\},\ x > 1.$$

Případ, kdy je $x < -1$, je podobný jako kdy je $x > 1$. Při této volbě x vyjde derivace $2x$. Tedy pro tuto volbu x bude zobecněný gradient:

$$\partial f(x) = conv\{2x\} = \{2x\},\ x < -1.$$

Jako další případ je, kdy $1 > x > -1$. Při této volbě x stačí pouze zderivovat funkci na daném intervalu. Protože je funkce konstatní, tak derivace vyjde nulová. Při této volbě x vyjde:

$$\partial f(x) = conv\{0\} = \{0\},\ x \in (-1, 1).$$

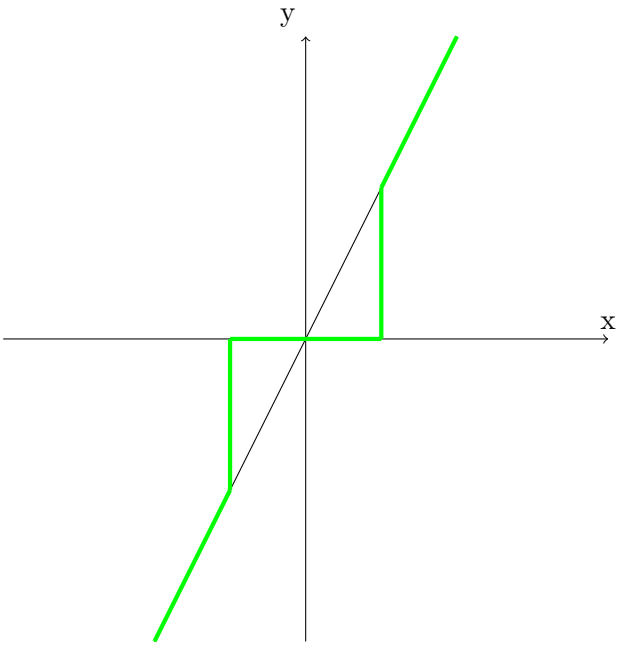
Jako poslední věc nám chybí zjistit zobecněný gradient pro $x = 1$ a $x = -1$. Začneme s volbou $x = 1$, kde zleva bude limita derivace nulová a zprava je $2x$, kde nám po dosazení tedy vyjde 2. Zobecněný gradient při $x = 1$ bude vypadat takhle:

$$\partial f(1) = conv\{0, 2\} = \langle 0, 2\rangle$$

a při volbě $x = -1$:

$$\partial f(-1) = conv\{0, -2\} = \langle 0, -2\rangle.$$

Zobecněný gradient funkce $f(x) = max\{x^2, 1\}$ bude vypadat takto:



b) Teď, když jsme schopni vypočítat zobecněný gradient, podíváme se jak se počítá zobecněný Jacobián.

   i) Jako první si spočteme zobecněný Jacobián této vektorové funkce:

$$F(x) = (|x_1| + |x_2|, |x_1|) \text{ v bodě } x = (0, 1).$$

U druhé souřadnice má tato vektorová funkce derivaci, tudíž nastává problém pouze v první souřadnici, protože v bodě 0 nemá zadaná funkce "klasickou"derivaci. Příklad si rozdělíme na dva případy a to kdy $F(x_1, x_2)$ rozdělíme na $x_1 \to 0_-, x_2 \to 1$ a $x_1 \to 0_+, x_2 \to 1$. Začneme tedy s možností, kdy $x_1 \to 0_+, x_2 \to 1$. x_1 konverguje k 0 zprava,

tudíž bude x_1 kladné a tedy $|x_1| = x_1$ a x_2, které konverguje zprava do bodu 1, bude také kladné. Tedy můžeme říci, že

$$F(x) = F(x_1 + x_2, x_1).$$

Když parciálně zderivujeme vektorovou funkci výše, tak nám vyjde tento Jacobián

$$JF(x_1, x_2) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}.$$

Analogicky budeme postupovat i u možnosti, kdy $x_1 \to 0_-, x_2 \to 1$. U x_2, které jde do 1, bude opět x_2 kladné, ale x_1, jdoucí do 0 zleva, tedy $|x_1| = -x_1$. Tedy platí, že

$$F(x) = F(-x_1 + x_2, -x_1)$$

a výsledný Jacobián bude vypadat takto

$$JF(x_1, x_2) = \begin{bmatrix} -1 & 1 \\ -1 & 0 \end{bmatrix}.$$

Výsledný zobecněný Jacobián dostaneme jako

$$\partial F(x) = conv\{\begin{bmatrix} a & 1 \\ b & 0 \end{bmatrix}, a \in \langle -1, 1\rangle\, b \in \langle -1, 1\rangle\}.$$

ii) Jako další budeme počítat zobecněný Jacobián u této vektorové funkce:

$$F(x) = (|x_1 + x_2|, |x_1 \cdot x_2|) \text{ v bodě } x = (0, 0).$$

U této funkce musíme zjistit Jacobián pro každý kvadrant. Navíc u 4. a 2. kvadrantu se nám případ větví na další dvě možnosti. Nejprve začneme s možností, kdy $x_1 \to 0_+, x_2 \to 0_+$. U této možnosti jsou x_1 i x_2 kladná, tudíž:

$$F(x) = (x_1 + x_2, x_1 \cdot x_2), \text{ kde } x_1 > 0 \text{ a } x_2 > 0.$$

Jacobián v bodě $x = (x_1, x_2)$ vyjde:

$$JF(x) = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

Dále budeme řešit případ, kdy $x_1 \to 0_-, x_2 \to 0_-$. Zde budou x_1 a x_2 záporná, tedy:

$$F(x) = (-x_1 - x_2, x_1 \cdot x_2).$$

Po parcialní derivaci v bodě $x = (x_1, x_2)$ vyjde:

$$JF(x) = \begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix}.$$

Nyní potřebujeme zjistit jak bude naše funkce vypadat, když $x_1 \to 0_+, x_2 \to 0_-$. Zde si případ rozdělíme ještě na případ, kdy $|x_1| \geq |x_2|$ a $|x_1| < |x_2|$. Když bude $|x_1| \geq |x_2|$, tak poté bude

$$F(x) = (x_1 + x_2, -x_1 \cdot x_2)$$

a derivace této funkce bude:

$$JF(x) = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}.$$

Pro možnost $|x_1| < |x_2|$ bude vypadat funkce

$$F(x) = (-x_1 - x_2, -x_1 \cdot x_2)$$

a derivace

$$JF(x) = \begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix}.$$

Nyní nám chybí už jen když $x_1 \to 0_-, x_2 \to 0_+$. Případ se opět větví a tentokrát na $|x_2| \geq |x_1|$ a $|x_2| < |x_1|$. Budeme postupovat analogicky jako u minulého případu. Tedy pro $|x_2| \geq |x_1|$ bude

$$F(x) = (x_1 + x_2, -x_1 \cdot x_2)$$

a

$$JF(x) = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

a pro $|x_2| < |x_1|$ bude

$$F(x) = (-x_1 - x_2, -x_1 \cdot x_2)$$

a

$$JF(x) = \begin{bmatrix} -1 & -1 \\ 0 & 0 \end{bmatrix}.$$

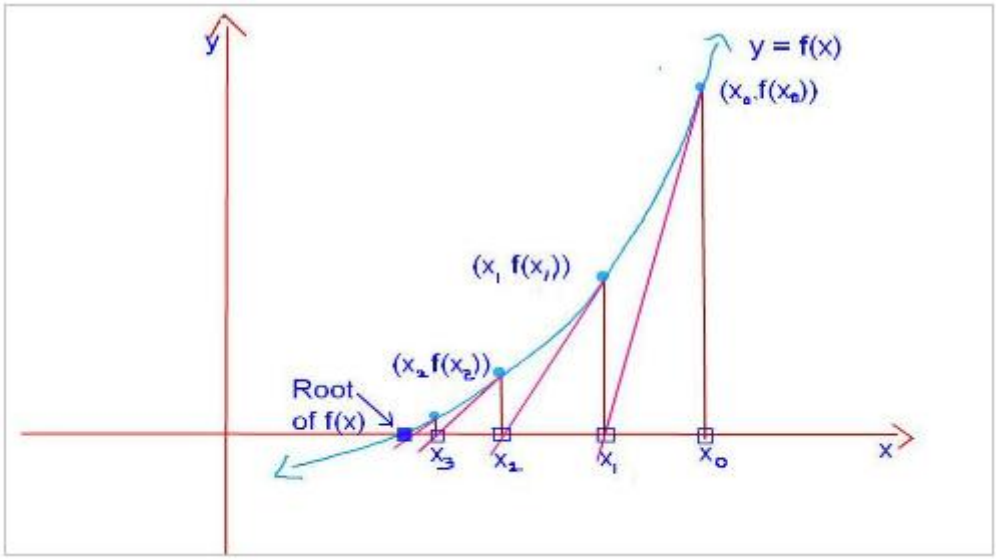
Výsledný zobecněný Jacobián bude tedy vypadat:

$$\partial F(x) = conv\{\begin{bmatrix} a & b \\ 0 & 0 \end{bmatrix}, a \in \langle -1, 1\rangle\, b \in \langle -1, 1\rangle\}.$$

3 Newtonova metoda

3.1 "Klasická"Newtonova metoda

Newtonově metodě se také říká metoda tečen, což plyne z geometrické interpretace:



Pomocí této metody se nalézají nulové body funkce. Při hledání nulových bodů funkce musí být daná funkce diferencovatelná ve všech bodech. Hledání nulového bodu znamená hledání bodu x^*, pro který platí

$$f(x^*) = 0$$

a dále se postupuje podle vzorce:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{9}$$

Iteracemi dojdeme k aproximaci požadovaného výsledku.
Při hledání nulových bodů u soustavy rovnic, musí být všechny funkce v soustavě diferencovatelné ve všech bodech. Hledání nulových bodů znamená hledání vektoru x^*, pro který platí

$$F(x^*) = o.$$

Ve vzorci (9) se použije místo první derivace Jacobián funkce F, a tedy vzorec bude vypadat následovně:

$$x_{k+1} = x_k - JF(x_k)^{-1} \cdot F(x_k). \tag{10}$$

Opět se iteracemi vzorce (10) dostaneme k požadované aproximaci výsledku.

3.2 Semihladká Newtonova metoda

Semihladká Newtonova metoda se použije pro řešení rovnice či soustavy rovnic, kdy funkce nejsou diferencovatelné ve všech bodech. Následující definice a věty jsem čerpal z [1], zde jsou uvedené i důkazy, které v mé práci neuvádím.

Definice 3.1 *Nechť $U \subset \mathbb{R}^n$ je neprázdná a otevřená množina. Funkce $F : U \to \mathbb{R}^m$ je semihladká v $x \in U$, pokud je lokálně lipschitzovsky spojitá v x a pokud*

$$\lim_{\substack{G \in \partial F(x+t\bar{d}) \\ \bar{d} \to d, t \downarrow 0}} G\bar{d} \tag{11}$$

existuje pro všechna $d \in \mathbb{R}^n$. Pokud je F semihladká pro všechna $x \in U$, tak nazýváme F semihladkou (na U).

Nejprve si zavedeme, co je superlinearita $\mathcal{o}$.

Definice 3.2 *Nechť $f, g : \mathbb{R}^n \to \mathbb{R}^m$ a $x^* \in \mathbb{R}^n$, poté píšeme, že $f(x) = \mathcal{o}(g(x))$ pro $x \to x^*$ když pro všechna $\epsilon > 0$ existuje okolí U od bodu x^*, tak že pro všechna $x \in U(x^*)$ platí:*

$$||f(x)|| \leq \epsilon ||g(x)||. \tag{12}$$

Poté platí:

Věta 3.1 *Nechť $F : U \to \mathbb{R}^m$ je definována na otevřené množině $U \subset \mathbb{R}^n$. Pak pro $x \in U$ jsou nasledující podmínky ekvivalentní:*

1. *F je semihladká v x.*
2. *F je lokálně lipschitzovsky spojitá v x, $F'(x;.)$ existuje, a pro jakékoliv $G \in \partial F(x+d)$ platí:*

$$||Gd - F'(x,d)|| = \mathcal{o}(||d||) \text{ a } d \to 0. \tag{13}$$

3. *F je lokálně lipschitzovsky spojitá v x, $F'(x;.)$ existuje a pro $G \in \partial F(x+d)$ platí:*

$$||F(x+d) - F(x) - Gd|| = \mathcal{o}(||d||) \text{ a } d \to 0. \tag{14}$$

Dále platí tvrzení:

Věta 3.2 *Nechť $U \subset \mathbb{R}^n$ je otevřená. Pokud $F : U \to \mathbb{R}^m$ je spojitě diferencovatelná v okolí $x \in U$, pak F je semihladká v x.*

Semihladká Newtonova metoda vychází ze vztahu (10), s tou vyjímkou že místo klasického Jacobiánu se použije zobecněný Jacobián z Clarkeova kalkulu. Vzorec k dosažení výsledku bude vypadat následovně:

$$x_{k+1} = x_k - \partial F(x_k)^{-1} \cdot F(x_k) \tag{15}$$

kde $\partial F(\cdot)$ značí zobecněný Jacobián, který je při počítání s jednou funkcí totožný se zobecněným gradientem. Dále potřebujeme vědět, zda funkce bude konvergovat. Nejprve si řekneme, kdy bude Newtonova metoda konvergovat lokálně.

Věta 3.3 *Předpokládejme, že $x^* \in \mathbb{R}^n$ splňuje $F(x^*) = o$, F je lipschitzovsky spojitá a semihladká v x^* a $\partial F(x^*)$ je regulární. Poté existuje $\epsilon > 0$, takové že pro $x_0 \in U(x^*, \epsilon)$ posloupnost $\{x_k\}$ generovaná pomocí semihladké Newtonovy metody je dobře definovaná, konverguje k x^*, a splňuje*

$$||x_{k+1} - x^*|| = \mathcal{O}(||x_k - x^*||),\ k \to +\infty \tag{16}$$

Ještě nám zbývá určit podmínky globální konvergence.

Věta 3.4 *Předpokládejme, že F je lokálně lipschitzovsky spojitá a semihladká na $U_0(x_0, r)$, které je uzávěrem $U(x_0, r)$. Také předpokládejme, že pro jakékoliv $G(x) \in \partial F(x)$, $x \in U_0$, $G(x)$ je regulární a pro $y \in U_0$ platí:*

$$||G(x)^{-1}|| \leq \beta,\ ||G(x)(y-x) - F'(x; y-x)|| \leq L||y-x||, \tag{17}$$

$$||F(y) - F(x) - F'(x; y-x)|| \leq \eta||y-x||, \tag{18}$$

kde $\alpha = \beta(L+\eta) < 1$ a $\beta||F(x_0)|| \leq r(1-\alpha)$. Poté iterace $\{x_k\}$ semihladké Newtonovy metody zůstávají v U_0 a konvergují k jedinému řešení x^* rovnice $F(x) = o$ v U_0. Navíc platí

$$||x_k - x^*|| \leq \frac{\alpha}{1-\alpha}||x_k - x_{k-1}||,\ k = 1, 2, \ldots \tag{19}$$

Úkoly, které řešíme v 5. kapitole obsahují funkce F, které jsou lokálně lipschitzovsky spojité a semihladké. Díky uvedeným větám tyto úlohy můžeme řešit semihladkou Newtonovou metodou a tato metoda konverguje k řešení.

4 Poznámky k implementaci semihladké Newtonovy metody

V této kapitole rozeberu implementaci semihladké Newtonovy metody pro řešení soustavy $F(x) = o$ v programu Matlab. Nejprve se budu zabývat výpočtem zobecněného Jacobiánu, poté naprogramováním kontroly Jacobiánu funkce $F(x)$ a jako poslední věc proberu implementaci samotné Newtonovy metody.

4.1 Výpočet zobecněného Jacobiánu

Nejprve ukážu kód pro funkci, kterou si nechávám generovat vektor funkčních hodnot. Takový kód může vypadat následovně.

```
function [a]=funcke(x)
    n=length(x);
    for i=1:n
        a(i)=abs(x(i)^2-i);
    end
    a=a';
```

Na vstupu je pouze vektor x a na výstupu je pouze vektor funkčních hodnot a. Nejprve si zjistím délku vektoru a poté procházím v cyklu for jednotlivé složky vektoru x, a aplikuju na danou hodnotu vzorec, který si dopředu určím. V tomhle případě aplikuji vzorec $|x_i^2 - i|$, kdy $i \in \mathbb{N}$. Aby mi vyšel sloupcový vektor, tak vektor a transponuji. Tuto funkci na generování funkčních hodnot používám v celé této kapitole.
Nyní můžu přistoupit k popisu výpočtu aproximace zobecněného Jacobiánu. U funkce počítající aproximaci zobecněného Jacobiánu bude na vstupu hodnota konstanty h, kterou používám při derivaci, a vektor, ke kterému se bude počítat derivace. Na výstupu bude výsledný Jacobián. Tedy:

```
function [JF]=GenJac(h,x)
```

dále si zjistíme velikost zadaného vektoru a zadefinujeme si Jacobián jako nulovou matici:

```
n=length(x);
JF=zeros(n,n);
```

Nyní můžeme přistoupit k samotnému počítání. Teď nejprve ukážu kód a poté ho vysvětlím:

```
beta=zeros(n,1);
for j=1:n
    beta(j)=1;
    JF(j,:)=(funcke(x+h*beta)-funcke(x))/h;
    beta=zeros(n,1);
end
```

V cyklu for procházím i-tou a j-tou složku v matici, kde zajišťuju parciální derivace pomocí vektoru beta, který vždy změním na nulový vektor. Do matice vkládám rovnou vektory, ke kterým

dospěju přes aproximovanou derivaci. Ve funkci "funkce"mám zadefinovanou danou vektorovou funkci. Poté je ještě třeba vypočítaný Jacobián transponovat, protože se mi vektory uložili do sloupců a ne do řádků. Výše uvedený výpočet je numerické řešení pomocí konečných diferencí. Kdyby jsme chtěli analytické řešení, tak nahradíme

```
JF(j,:)=(funcke(x+h*beta)-funcke(x))/h;
```

například následující syntaxí.

```
df=diff(abs(y)-1);
JF(j,:)=subs(df,'y',x(j))*beta;
```

Analytické řešení není napsané obecně. Musí se tedy vždy zadefinovat funkce, pro kterou se má vypočítat derivace, v případě výše je to funkce $|y| - 1$. Výsledný kód pro numerické řešení poté bude vypadat takto:

```
function [JF]=GenJac(h,x)
n=length(x);
JF=zeros(n,n);
beta=zeros(n,1);
for j=1:n
    beta(j)=1;
    JF(j,:)=(funcke(x+h*beta)-funcke(x))/h;
    beta=zeros(n,1);
end
JF=JF';
```

4.2 Kontrola funkce

Vstup pro kontrolní funkci bude matice A, alfa(pro kontrolu lipschitzovské spojitosti), x(vektor), h(hodnota o kterou budeme zvyšovat vektor). Hlavně potřebujeme zjistit zda je matice regulární. To zkontrolujeme pomocí if:

```
if det(A)>0 || det(A)<0
```

Pro kontrolu lipsichtzovské spojitosti opět použijeme if:

```
if norm(funcke(x)-funcke(xk))<=(alfa*norm(x-xk))
```

Hodnotu xk si spočteme jako $xk = x + h$. Poté se jen dopíšou chybová hlášení. Výsledný kód může vypadat takto:

```
function Control(A,alfa,x,h)
xk=x+h;
if det(A)>0 || det(A)<0
    if norm(funcke(x)-funcke(xk))<=(alfa*norm(x-xk))
        disp('Funkce je v poradku.');
    else
        disp('Funkce neni lipschitzovsky spojita.');
    end
else
    error('Jacobian neni regularni matici.');
end
```

4.3 Implementace semihladké Newtonovy metody

Pro následující funkci bude vypadat zápis následovně.

```
function [x,k]=NewtonMethod(x0,presnost)
```

Na vstupu pro tuto funkci bude bod x_0 a přesnost. Nejprve je nutné si vypočítat počáteční bod. Zvolíme si první bod jako bod $(0, 0, 0)$ a dále budeme dosazovat do nám známého vzorce:

```
b=funcke(x0);
A=GenJac(0.01,x0);
x=x0-A\b;
xk=x0;
```

Poté si Jacobián zkontrolujeme naší kontrolní funkcí Control a nastavíme čítač iterací na 1:

```
Control(A,0.5,x,0.1);
k=1;
```

Již nám pouze zbývá zadat cyklus v kterém to budeme počítat:

```
while norm(x-xk)>=presnost
    b=funcke(x);
    A=GenJac(0.01,x);
    xk=x;
    x=xk-A\b;
    k=k+1;
end
```

Při kontrole, zda bude cyklus pokračovat, se kontroluje přesnost. Dále se dosazuje do klasického vzorce pro Newtonovu metodu a při každém kroku přičtu k čítači iterací 1. Výsledný kód pro výpočet Newtonovy metody bude vypadat takto:

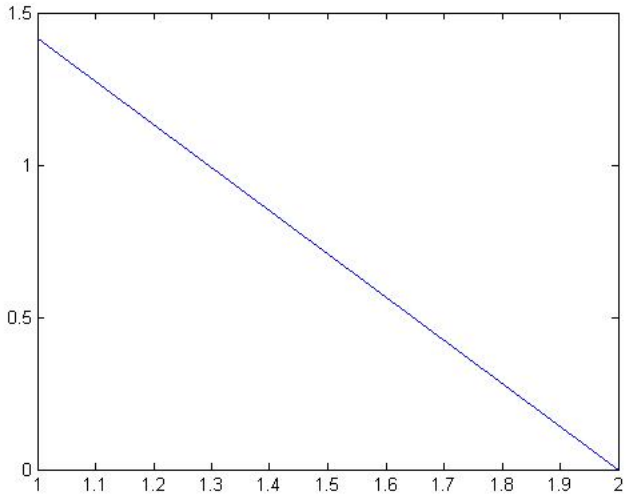
```
function  [x,k]=NewtonMethod(x0,presnost)
    b=funcke(x0);
    A=GenJac(0.01,x0);
    x=x0-A\b;
    xk=x0;
    Control(A,0.5,x,0.1) ;
    k=1;
    while norm(x-xk)>=presnost
        b=funcke(x);
        A=GenJac(0.01,x);
        xk=x;
        x=xk-A\b;
        k=k+1;
    end
```

5 Numerické testování

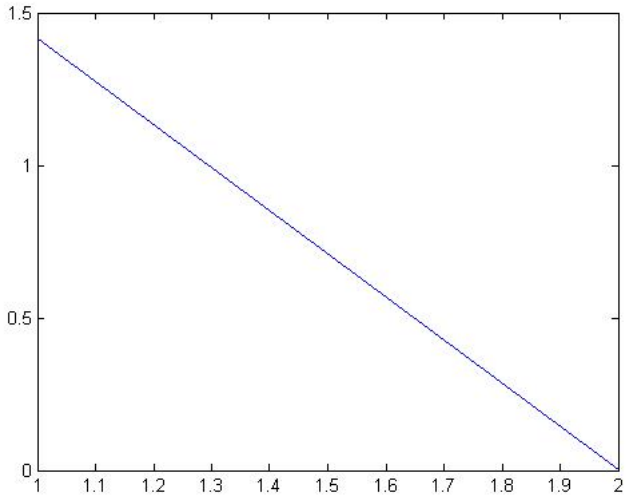
5.1 Srovnání řešení pomocí analyticky vypočteného Jacobiánu a numericky vypočteného Jacobiánu

Pro srovnání jsem si vybral tyto vektorové funkce: $F_1(x) = (|x_1|, |x_2|, |x_3|)$, $F_2(x) = (|x_1 - 1|, |x_2 - 2|, |x_3 - 3|)$, $F_3(x) = (|x_1^2 - 1|, |x_2^2 - 2|, |x_3^2 - 3|)$. Počítat budu s přesnosti na 10^{-7} s počátečním bodem $[1, 0, 1]$. Pro představu, jak se k řešení konvergovalo, jsem nechal zakreslit do grafu, jak se měnila norma funkčních hodnot $||f||$ v každé iteraci. Na y-ové ose bude velikost $||f||$ a na x-ové ose bude jaké iteraci daná hodnota odpovídá.

1. V prvním případě řeším soustavu $|x_1| = 0$, $|x_2| = 0$ a $|x_3| = 0$. Levá strana soustavy se dá zapsat jako vektorová funkce $F_1(x) = (|x_1|, |x_2|, |x_3|)$ a chceme vyřešit problém, kdy $F_1(x) = o$. V tomto případě vyšel výsledek $x = [0, 0, 0]$. Při využití numericky spočteného Jacobiánu i při využití analyticky spočteného Jacobiánu se k řešení došlo po dvou iteracích.

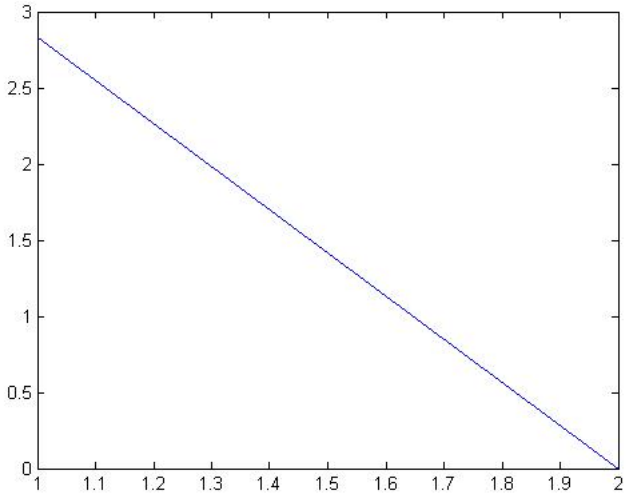


Obrázek 1: Řešení $F_1(x) = o$ pomocí analyticky spočteného Jacobiánu

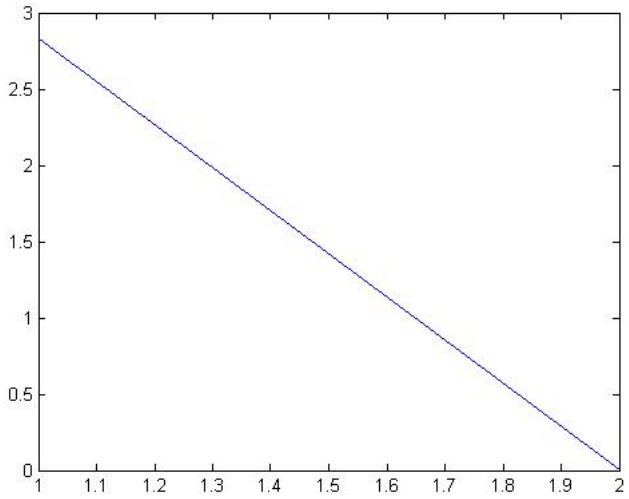


Obrázek 2: Řešení $F_1(x) = o$ pomocí numericky spočteného Jacobiánu

2. V druhém případě řeším soustavu $|x_1 - 1| = 0$, $|x_2 - 2| = 0$ a $|x_3 - 3| = 0$. Levá strana soustavy se dá opět zapsat jako vektorová funkce $F_2(x) = (|x_1-1|, |x_2-2|, |x_3-3|)$. Chceme vyřešit problém, kdy $F_2(x) = o$. U druhého případu tedy vyšel výsledek $x = [1, 2, 3]$. Při využívání analyticky spočteného Jacobiánu, tak i numericky spočteného Jacobiánu se k výsledku došlo po dvou iteracích.

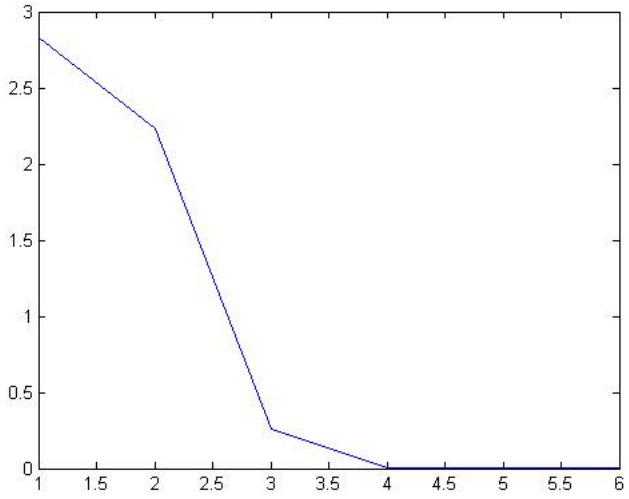


Obrázek 3: Řešení $F_2(x) = o$ pomocí analyticky spočteného Jacobiánu

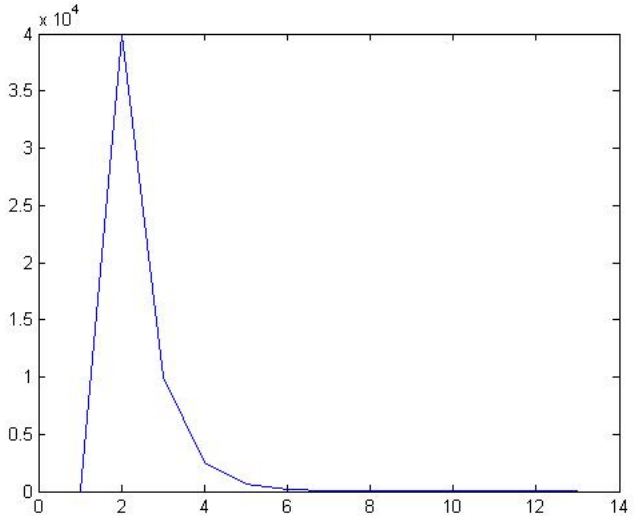


Obrázek 4: Řešení $F_2(x) = o$ pomocí numericky spočteného Jacobiánu

3. V posledním případě řeším soustavu $|x_1^2 - 1| = 0$, $|x_2^2 - 2| = 0$ a $|x_3^2 - 3| = 0$. I zde se dá levá strana soustavy zapsat jako vektorová funkce $F_3(x) = (|x_1^2 - 1|, |x_2^2 - 2|, |x_3^2 - 3|)$. Chceme opět vyřešit problém, kdy $F_3(x) = o$. V posledním případě vyšel výsledek $x = [1, 1.4142, 1.7321]$. Při využití analyticky spočteného Jacobiánu se k výsledku došlo po 6 iteracích a při využití numericky spočteného Jacobiánu se došlo k výsledku po 13 iteracích.



Obrázek 5: Řešení $F_3(x) = o$ pomocí analyticky spočteného Jacobiánu



Obrázek 6: Řešení $F_3(x) = o$ pomocí numericky spočteného Jacobiánu

Pro lepší srovnání jsem to zapsal do tabulky. V prvním sloupci jsou problémy, které jsem řešil, v druhém sloupci je počet iterací při využití analyticky spočteného Jacobiánu a ve třetím sloupci je počet iterací při využití numericky spočteného Jacobiánu.

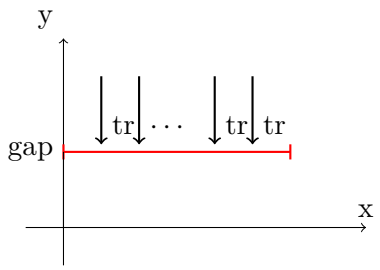
Vektorová funkce	počet iterací s použitím analyticky spočteným Jacobiánem	počet iterací s použitím numericky spočteným Jacobiánem
$F_1(x) = o$	2	2
$F_2(x) = o$	2	2
$F_3(x) = o$	6	13

Můžeme sledovat, že řešení s analyticky spočteným Jacobiánem není nikdy pomalejší než řešení s numericky spočteným Jacobiánem a ve třetím případě je i dle očekávání rychlejší.

5.2 Praktická úloha

5.2.1 Teoretický základ

Jako praktickou úlohu hodlám řešit deformaci struny při kontaktu s překážkou. Červená usečka



označuje pružnou strunu délky l, na kterou působí vnější síla, jejíž délková hustota je tr. Ve vzdálenosti gap se nachází pod strunou tuhá překážka (její hranice je osa x). Zmíněnou úlohu budeme řešit numericky, kdy si strunu diskretizujeme a deformaci vypočteme jen v bodech diskretizace (jejich počet je n). K numerickému řešení použijeme metodu konečných prvků [5]. Nejprve využiju metodu konečných prvků k sestavení matice tuhosti K řádu n

$$K = \begin{bmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ 0 & 2 & -1 & 0 & \dots & 0 \\ \vdots & -1 & 2 & \ddots & \ddots & \vdots \\ \vdots & 0 & \ddots & \ddots & -1 & 0 \\ \vdots & \vdots & \ddots & -1 & 2 & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix}$$

Poté vypočteme vektor síly f řádu n.

$$f = \begin{bmatrix} 0 \\ tr \\ \vdots \\ tr \\ 0 \end{bmatrix} \cdot h^2, \text{ kde } h = \frac{l}{n}.$$

Výše zmíněná úloha může být popsána jako řešení následující úlohy:

$$\begin{gathered} Ku + \lambda = f \\ u - gap \geq 0, \lambda \geq 0, \lambda(u - gap) = 0, \end{gathered} \tag{20}$$

kde u popisuje prohnutí struny a λ napětí mezi strunou a překážkou a gap je vzdálenost překážky od struny. Tuto úlohu lze přeformulovat na:

$$Ku + \lambda = f \tag{21}$$

$$\lambda - max(0, \lambda - \rho(u - gap)) = 0, \rho > 0. \tag{22}$$

ρ volím v našich úlohách na pevno a to hodnotu 11. Tato soustava obsahuje u druhé rovnice nediferencovatelnou funkci, a proto musím použít semihladkou Newtonovou metodu. Výše uvedenou soustavu rovnic jsem si přepsal na problém s neznámou x:

$$Kx(1:n) + x(n+1:2n) - f = 0$$
$$x(n+1:2n) - max(0, x(n+1:2n) - \rho(x(1:n) - gap)) = 0,$$

kde $x(1:n)$ se rovná vektoru prohnutí u a $x(n+1:2n)$ se rovná vektoru napětí λ. Funkční hodnota $F(x)$ bude poté vypadat takto:

$$F(x) = \begin{bmatrix} K_1x + L_1x - f \\ L_1x - max(0, L_1x - \rho(L_2x - gap)) \end{bmatrix}$$

zde:

$$K_1 = \begin{bmatrix} K & O \end{bmatrix}$$

$$L_1 = \begin{bmatrix} O & I \end{bmatrix}$$

$$L_2 = \begin{bmatrix} I & O \end{bmatrix}.$$

Množina popisující zobecněný Jacobián bude poté vypadat takto:

$$\begin{bmatrix} K_1 + L_1 \\ L_1 - max(0, L_1 - \rho L_2) \end{bmatrix} \in \partial F(x).$$

Tohle jsem naprogramoval a popsal v následující podkapitole a využil k výpočtům. Nakonec se dá tato úloha napsat jako minimalizační úloha:

$$min \frac{1}{2} u^T K u - u^T f \tag{23}$$
$$u \leq gap.$$

protože má Matlab již předdefinovanou funkci pro výpočet minimalizační úlohy, tak jsem ji použil pro kontrolu.

5.2.2 Implementace

Nejprve jsem si vytvořil funkci GenMatrix, pro generování matice tuhosti a vektoru sil.

```
function [K,f]=GenMatrix(n,tr)
    K=zeros(n,n);
    f=ones(1,n)*tr;
    f(1)=0;
    f(n)=0;
    K(1,1)=1;
    K(2,2)=2;
    K(2,3)=-1;
    K(n,n)=1;
    K(n-1,n-1)=2;
    K(n-1,n-2)=-1;
    for i=3:n-2
        K(i,i)=2;
        K(i,i-1)=-1;
        K(i,i+1)=-1;
    end
```

V této funkci určuji velikost síly pomocí tr a očekávám, že se bude působit po celé délce kromě konců, kde bude nulová síla, stejnou silou.
Dále jsem naprogramoval funkci pro výpočet Jacobiánu SMJacobian. Tuto funkci rozeberu po částech.

```
function [JF]=SMJacobian(K,x,ro,g)
    n=length(x)/2;
    gap=ones(n,1)*g;
    K1=horzcat(K,zeros(n,n));
    L1=horzcat(zeros(n,n),eye(n,n));
    L2=horzcat(eye(n,n),zeros(n,n));
```

Do této funkce se zadá matice tuhosti K, vektor x, ve kterém se bude počítat Jacobián, ro, které značí hodnotu ρ a g, které značí vzdálenost překážky od osy. Z x si zjistíme velikost jednoho vektoru a z g vypočteme vektor *gap*. Dále si připravíme matice K1, L1, L2.

```
V=(K1+L1);
JF=vertcat(V,zeros(n,2*n));
```

První část Jacobiánu je jasná, je to matice V, která je součtem matic K1 a L1. Dále si poté nainicializujem matici JF, která značí náš hledaný Jacobián.

```
v=(L1*x+ro*(L2*x-gap));
for i=1:n
    if v(i)>0
    JF(n+i,:)=-ro*L2(i,:);
    else
    JF(n+i,:)=L1(i,:);
    end
end
```

Druhou část Jacobiánu už budeme muset zjišťovat řádek po řádku a proto využijeme cyklus for. Nejprve si spočteme vektor v, který má velikost n a budeme zjišťovat, kdy je větší i-tá složka větší než nula a podle toho vložíme vektor do Jacobiánu. Celá funkce pro výpočet Jacobiánu bude tedy vypadat takto:

```
function [JF]=SMJacobian(K,x,ro,g)
    n=length(x)/2;
    gap=ones(n,1)*g;
    K1=horzcat(K,zeros(n,n));
    L1=horzcat(zeros(n,n),eye(n,n));
    L2=horzcat(eye(n,n),zeros(n,n));
    V=(K1+L1);
    JF=vertcat(V,zeros(n,2*n));
    v=(L1*x+ro*(L2*x-gap));
    for i=1:n
        if v(i)>0
            JF(n+i,:)=-ro*L2(i,:);
        else
            JF(n+i,:)=L1(i,:);
        end
    end
```

Vytvořil jsem si funkci SpecialMethod pro výpočet úlohy $F(x) = 0$.

```
function [x, it]=SpecialMethod(K,f,ro,g,delka,presnost)
```

kde K je matice tuhosti, f vektor síly, ro hodnota ρ, g vzdálenost překážky od osy, delka je délka struny, presnost je přesnost na jakou bude počítat Newtonova metoda.

```
n=length(f);
h=delka/n;
x=zeros(2*n,1);
gap=ones(n,1)*g;
F=zeros(2*n,1);
```

Z vektoru sil zjistím velikost diskretizace, díky které poté vypočítám krok h. Poté si připravím neznámý vektor x o velikosti 2n a vektor gap, který značí vzdálenost k překážce, a nakonec inicializuji vektor funkčních hodnot F.

```
K1=horzcat(K,zeros(n,n));
L1=horzcat(zeros(n,n),eye(n,n));
L2=horzcat(eye(n,n),zeros(n,n));
```

Tímto kódem si připravím matice K1, L1 a L2.

```
F(1:n,1)=K1*x+L1*x-f;
v=(L1*x+ro*(L2*x-gap));
p1=(L2*x-gap);
p2=(L1*x);
for i=1:n
    if v(i)>0
        F(n+i,1)=-ro*p1(i);
    else
        F(n+i,1)=p2(i);
    end
end
```

Zde vypočítávám funkční hodnoty F k vektoru x. První polovina funkčních hodnot se určí jednoznačně, ale druhou polovinu musíme procházet v cyklu a určovat pro každý prvek zvlášť podle toho, zda je v větší než nula.

```
xk=x;
x=xk-(SMJacobian(K,x,ro,g)\F);
it=1;
```

Zde si vypočteme první iteraci Newtonovy metody a inicalizujeme čítač iterací na 1.

```
while norm(F)>=presnost
    F(1:n,1)=K1*x+L1*x-f;
    v=(L1*x+ro*(L2*x-gap));
    p1=(L2*x-gap);
    p2=(L1*x);
    for i=1:n
        if v(i)>0
            F(n+i,1)=-ro*p1(i);
        else
            F(n+i,1)=p2(i);
        end
    end
    xk=x;
    JF=SMJacobian(K,x,ro,g);
    x=xk-(JF\F);
    it=it+1;
end
```

Poté už jen provádíme výpočty funkčních hodnot a jednotlivých iterací v cyklu while, v kterém kontroluji přesnost podle funkční hodnoty. Celý kód poté vypadá následovně.

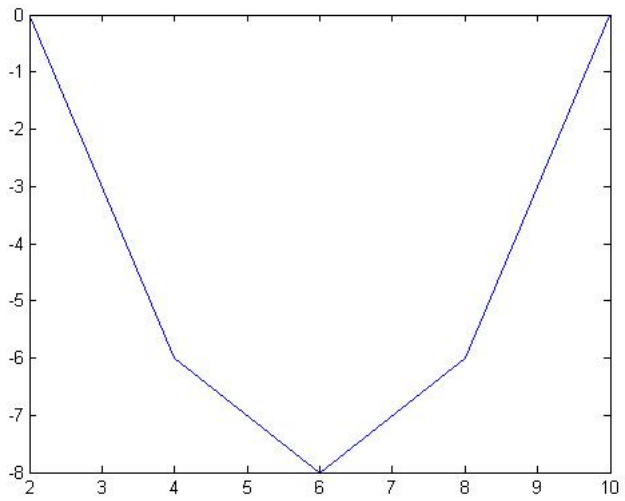
```
function [x, it]=SpecialMethod(K,f,ro,g,delka,presnost)
    n=length(f);
    h=delka/n;
    x=zeros(2*n,1);
    gap=ones(n,1)*g;
    F=zeros(2*n,1);

    K1=horzcat(K,zeros(n,n));
    L1=horzcat(zeros(n,n),eye(n,n));
    L2=horzcat(eye(n,n),zeros(n,n));

    F(1:n,1)=K1*x+L1*x-f;
    v=(L1*x+ro*(L2*x-gap));
    p1=(L2*x-gap);
    p2=(L1*x);
    for i=1:n
        if v(i)>0
            F(n+i,1)=-ro*p1(i);
        else
            F(n+i,1)=p2(i);
        end
    end
    xk=x;
    x=xk-(SMJacobian(K,x,ro,g)\F);
    it=1;
    while norm(F)>=presnost
        F(1:n,1)=K1*x+L1*x-f;
        v=(L1*x+ro*(L2*x-gap));
        p1=(L2*x-gap);
        p2=(L1*x);
        for i=1:n
            if v(i)>0
                F(n+i,1)=-ro*p1(i);
            else
                F(n+i,1)=p2(i);
            end
        end
        xk=x;
        JF=SMJacobian(K,x,ro,g);
        x=xk-(JF\F);
        it=it+1;
    end
```

5.2.3 Numerické testy

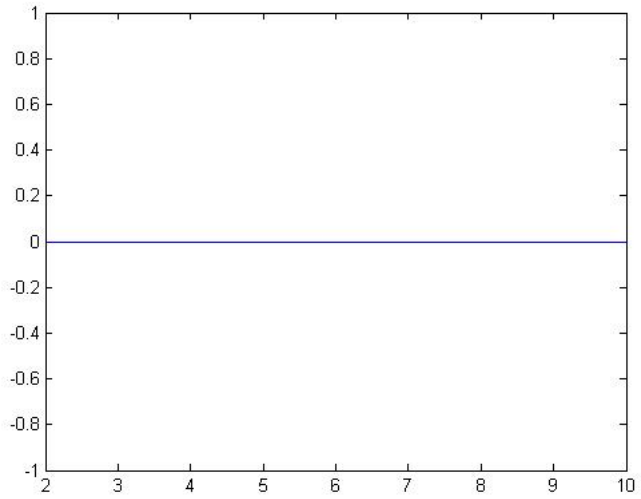
Numerické testy provedu pro různé velikosti matice tuhosti, pro různé délky struny a pro různé velikosti síly. Při tom $\rho = 11$, $gap = 250.1$. Počítat se bude s přesností 10^{-10}. U obrázků budu značit matici $K \in \mathbb{R}^{n \times n}$ jako $K[n, n]$. U každého testu je na grafu zaznamenán buď vektor prohnutí u nebo vektor napětí λ, s tím že na y-ové ose jsou hodnoty, které daný vektor nabýval, a na x-ové ose je délka struny.

1. Nyní provedu testy při působení na strunu silou o hustotě 1.

   (a) Nejprve si vygeneruju matici tuhosti řádu 5 a budu volit různé délky.
   Začnu s délkou struny 10, kdy vektor prohnutí u vyjde $[0, 6, 8, 6, 0]$ a vektor napětí λ vyjde $[0, 0, 0, 0, 0]$. Výsledek dostaneme po dvou iteracích. Při kontrole funkcí fmincon se dospělo k výsledku po sedmi iteracích a vektor prohnutí u vyšel $[0, 9.375, 12.5, 9, 375, 0]$, vektor napětí λ vyšel $[0, 0, 0, 0, 0]$. U grafů můžeme sledovat, že se struna neprohla dostatečně, aby se dotkla překážky, či dokonce aby se vytvořilo nějaké napětí. Určitá neshodnost výsledků byla nejspíš způsobena malým řádem matice K.

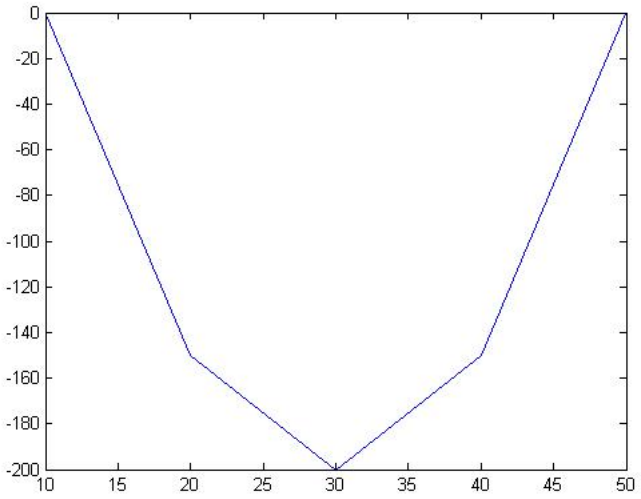


Obrázek 7: Vektor u k úloze $F(x) = 0$, tr=1, K[5,5], s délkou 10

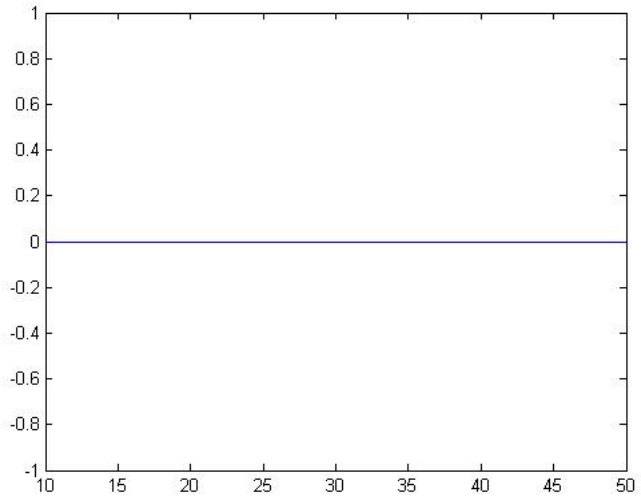


Obrázek 8: Vektor λ k úloze $F(x) = 0$, tr=1, K[5,5], s délkou 10

Dále zvolím délku struny 50. Vektor prohnutí u vyjde $[0, 150, 200, 150, 0]$ a vektor napětí λ vyjde $[0, 0, 0, 0, 0]$. Zde výsledek dostaneme po třech iteracích. Při použití funkce fmincon, vektor prohnutí u vyjde $[0, 203.175, 250.1, 203.175, 0]$ a vektor napětí λ vyjde $[0, 0, 62.4, 0, 0]$ po pěti iteracích. Při počítání pomocí semihladké Newtonovy metody se struna neprohla natolik, aby se dotkla překážky, ale při počítání s funkcí fmincon se struna překážky dotkla. Tento rozdíl byl způsoben nejspíš opět malým řádem matice tuhosti K.

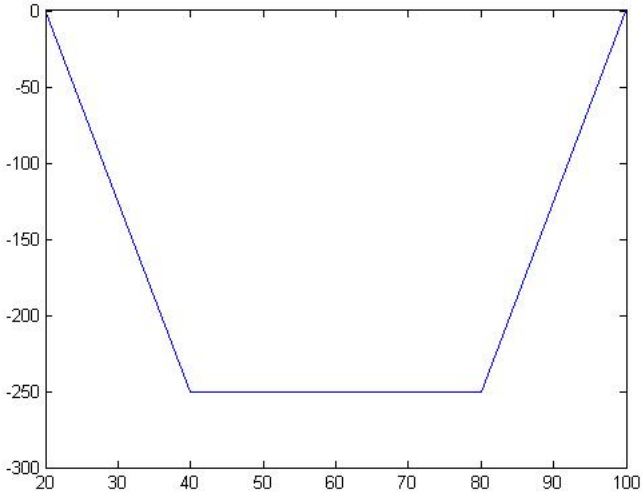


Obrázek 9: Vektor u k úloze $F(x) = 0$, tr=1, K[5,5], s délkou 50

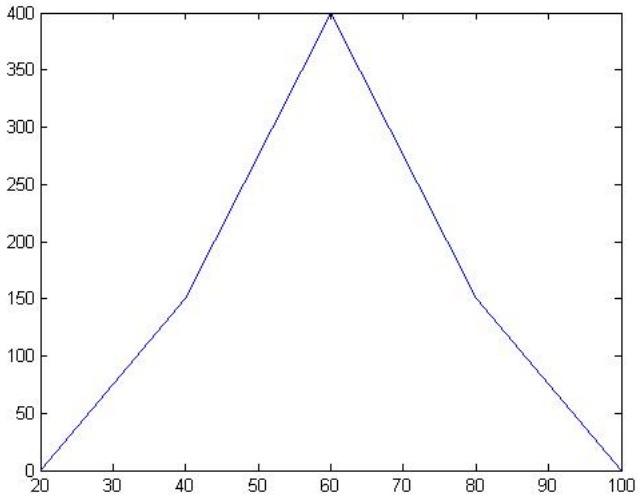


Obrázek 10: Vektor λ k úloze $F(x) = 0$, tr=1, K[5,5], s délkou 50

Nyní zvolím délku l 100. Vektor prohnutí u vyjde $[0, 250.1, 250.1, 250.1, 0]$ a vektor napětí λ vyjde $[0, 149.9, 400, 149.9, 0]$. Pro tyto volby opět dostaneme výsledek po třech iteracích. S použitím funkce fmincon se k výsledku dospělo po třech iteracích, kdy vektor prohnutí u vyjde $[0, 250.1, 250.1, 250.1, 0]$ a vektor napětí λ vyjde $[0, 374.9, 625, 374.9, 0]$. Stejně jako u příkladů výše se výsledky liší, nejspíš ze stejného důvodu. Při tomhle případě se výsledky liší pouze ve vektoru napětí λ.

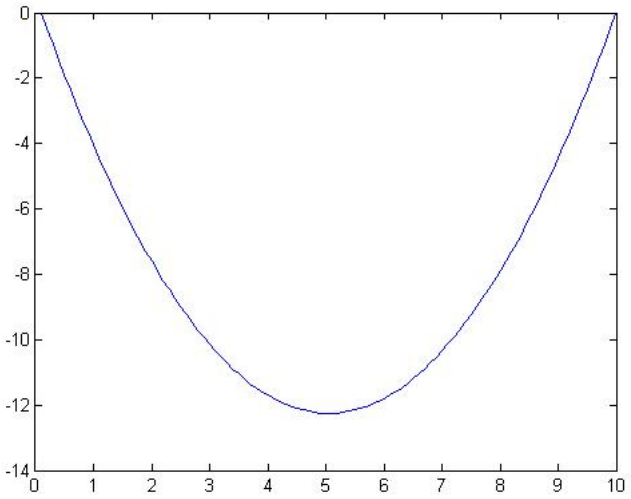


Obrázek 11: Vektor u k úloze $F(x) = 0$, tr=1, K[5,5], s délkou 100

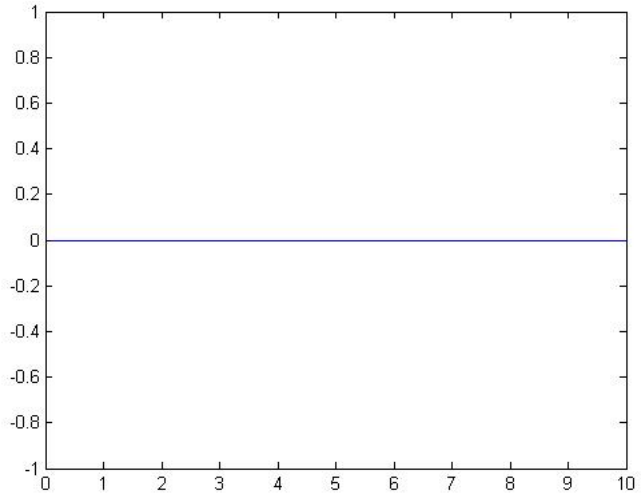


Obrázek 12: Vektor λ k úloze $F(x) = 0$, tr=1, K[5,5], s délkou 100

(b) Nyní si vygeneruju matici tuhosti řádu sto. Opět budu volit délku 10, 50 a 100. Zde již nebudu uvádět celý vektor kvůli jeho délky.
Pro délku struny 10 se ve vektoru prohnutí u objevuje nejvyšší hodnota 12,25 a vektor napětí λ je nulový. K tomuto výsledku se došlo po dvou iteracích. S použitím fmincon se dospělo k výsledku po 56-ti iteracích, kdy u vektoru prohnutí u vyšla nejvyšší hodnota 12.4985 a tedy vektor napětí λ vyšel nulový vektor. Nyní se výsledky liší minimálně a u obou možností výpočtu se struna nedotkla překážky. U tohohle případu vidíme, že tedy u úloh výše byla způsobena určitá nesrovnalost výsledků nejspíš příliš malým řádem matice K.

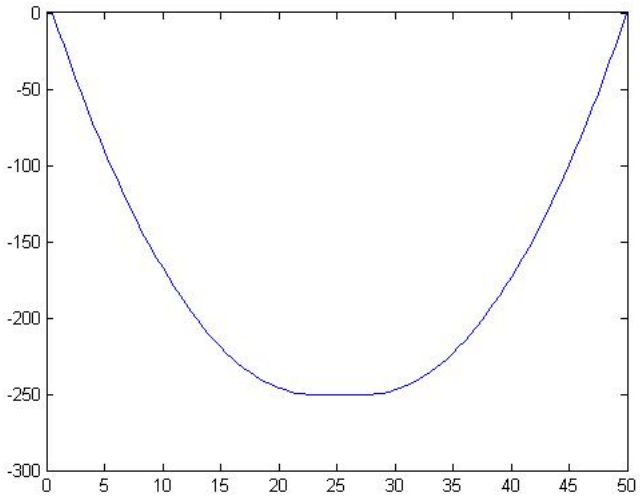


Obrázek 13: Vektor u k úloze $F(x) = 0$, tr=1, K[100,100], s délkou 10

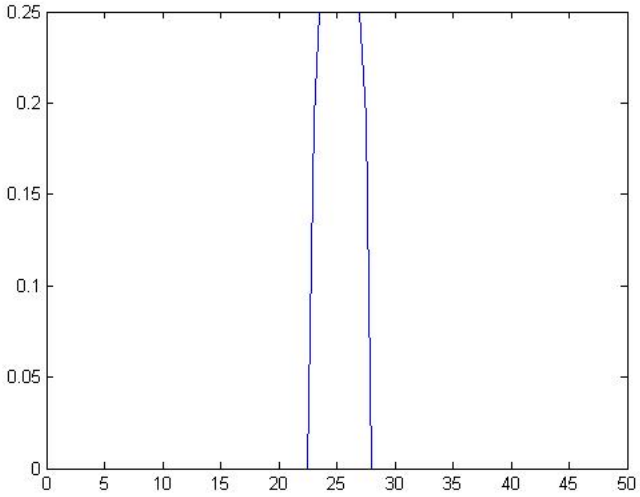


Obrázek 14: Vektor λ k úloze $F(x) = 0$, tr=1, K[100,100], s délkou 10

Pro délku 50 se ve vektoru prohnutí u objevuje nejvyšší hodnota 250,1 a u vektoru napětí λ je nejvyšší hodnota 0,25. K tomuto výsledku se došlo po 19-ti iteracích. Při využití funkce fmincon se došlo k výsledku po 53 iteracích, struna se dotkla překážky a u vektoru napětí λ vyšla největší hodnota 0,2551. V tomhle případě se taktéž výsledky, které byly vypočteny semihladkou Newtonovou metodou a pomocí funkce fmincon, moc nelišily. Struna se v tomhle případě dotkla překážky.

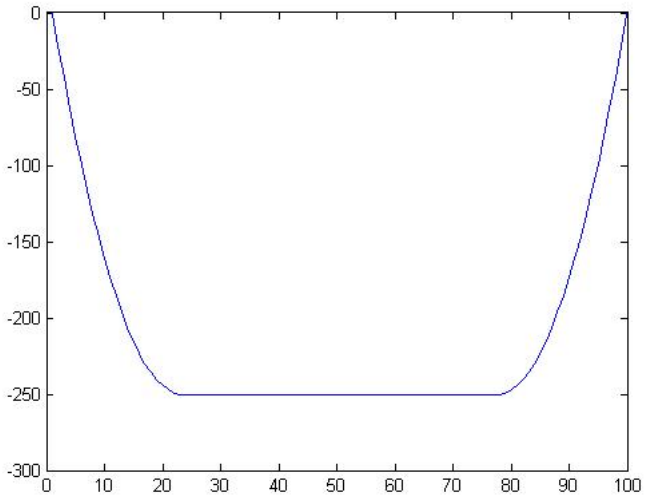


Obrázek 15: Vektor u k úloze $F(x) = 0$, tr=1, K[100,100], s délkou 50

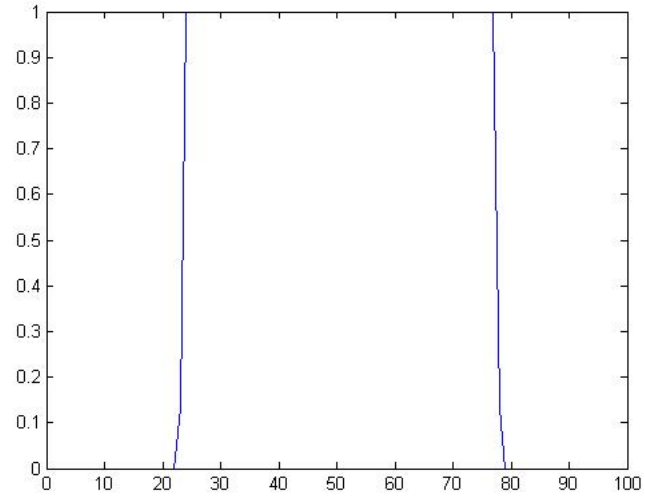


Obrázek 16: Vektor λ k úloze $F(x) = 0$, tr=1, K[100,100], s délkou 50

Pro délku 100 se opět ve vektoru prohnutí u objevuje nejvyšší hodnota 250,1 a u vektoru napětí λ je nejvyšší hodnota 1. K tomuto výsledku se došlo po 19-ti iteracích. Funkce fmincon dospěla k výsledku po 33 iteracích, struna i při tomhle výpočtu dotkla překážky a u vektoru napětí λ vychází nejvyšší hodnota 1,0203. Na tomto případě opět můžeme vidět, že výpočet semihladkou Newtonovou metodou je rychlejší než výpočet pomocí minimalizační úlohy. Struna se prohla až k překážce, ale způsobila malé napětí.



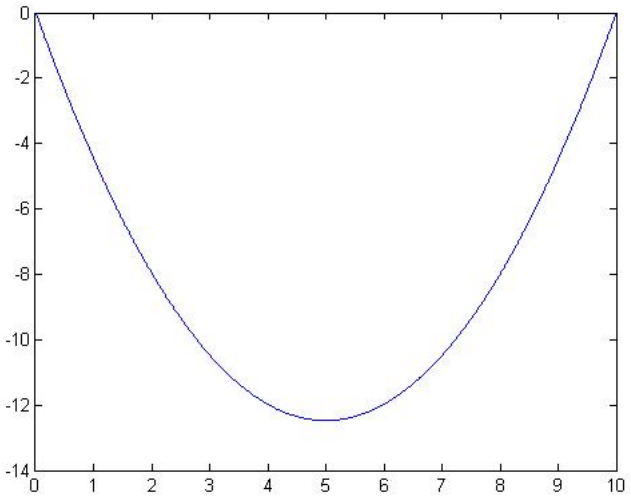
Obrázek 17: Vektor u k úloze $F(x) = 0$, tr=1, K[100,100], s délkou 100



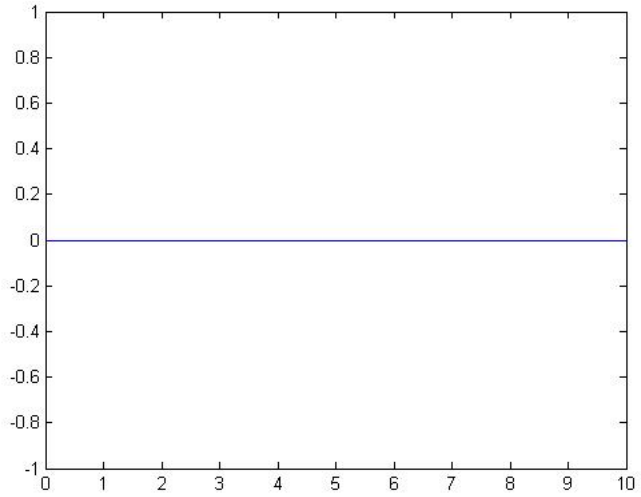
Obrázek 18: Vektor λ k úloze $F(x) = 0$, tr=1, K[100,100], s délkou 100

(c) Teď si vygeneruji matici tuhosti řádu 1000. Opět budu provádět testy na délkách 10, 50, 100.

Pro délku 10 se ve vektoru prohnutí u objevuje nejvyšší hodnota 12,45 a vektor napětí λ je nulový. K tomuto výsledku se došlo po dvou iteracích. Při využití funkce fmincon se dospělo k řešení po 507 iteracích a u vektoru napětí u byla největší hodnota 12,4807 a tedy vektor napětí λ vyšel nulový. Struna se tedy neprohla až k překážce.

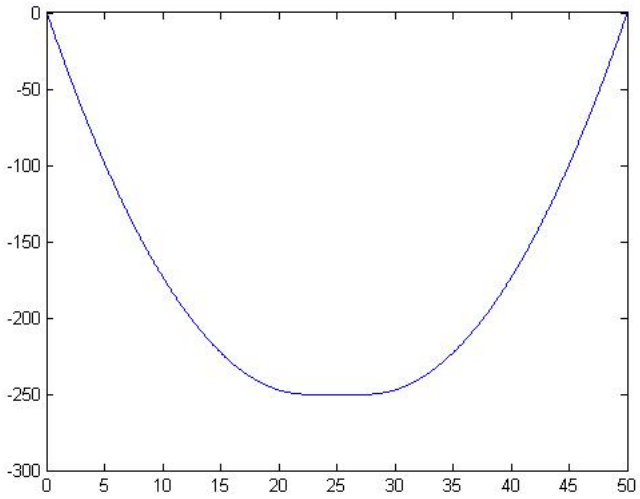


Obrázek 19: Vektor u k úloze $F(x) = 0$, tr=1, K[1000,1000], s délkou 10

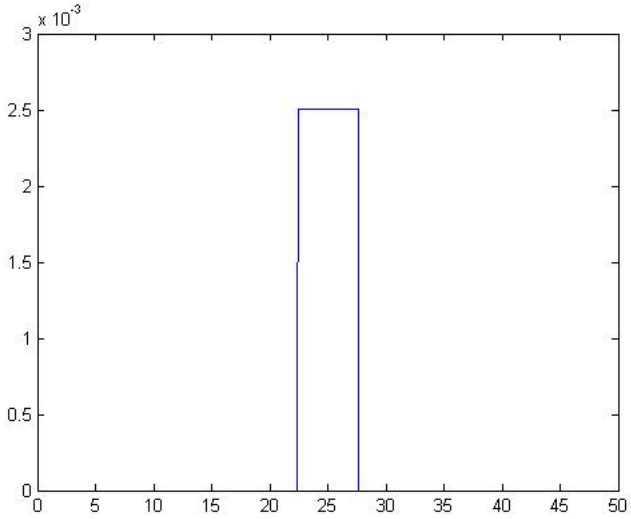


Obrázek 20: Vektor λ k úloze $F(x) = 0$, tr=1, K[1000,1000], s délkou 10

Pro délku 50 se ve vektoru prohnutí u objevuje nejvyšší hodnota 250,1 a u vektoru napětí λ je nejvyšší hodnota 0,0025. K tomuto výsledku se došlo po 172 iteracích. Při použití funkce fmincon se u vektoru prohnutí u objevila nejvyšší hodnota 250.1, u vektoru napětí λ se objevila nejvyšší hodnota 0.0025 a dospělo se k výsledku po 459 iteracích. Struna se prohla až k překážce, ale pouze se dotkla a způsobila pouze zanedbatelné napětí.

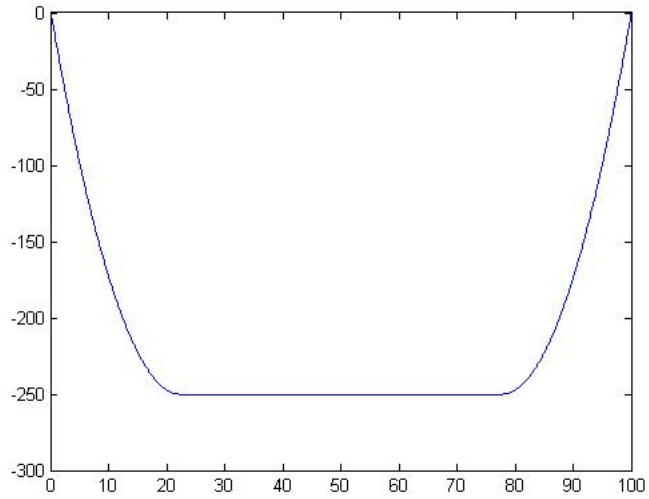


Obrázek 21: Vektor u k úloze $F(x) = 0$, tr=1, K[1000,1000], s délkou 50

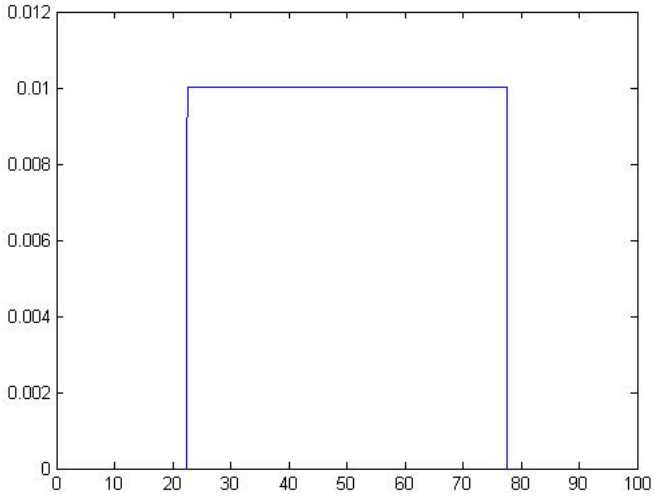


Obrázek 22: Vektor λ k úloze $F(x) = 0$, tr=1, K[1000,1000], s délkou 50

Pro délku 100 se opět ve vektoru prohnutí u objevuje nejvyšší hodnota 250,1 a u vektoru napětí λ je nejvyšší hodnota 0,01. K tomuto výsledku se došlo po 174 iteracích. Při řešení pomocí funkce fmincon se dospělo k řešení po 232 iteracích. Struna se prohla až k překážce, tedy vektor prohnutí u má nejvyšší hodnotu 250.1 a vektor napětí λ má nejvyšší hodnotu 0,01.

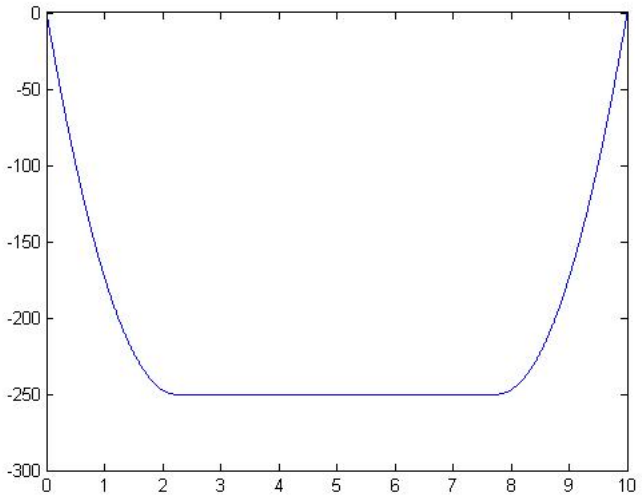


Obrázek 23: Vektor u k úloze $F(x) = 0$, tr=1, K[1000,1000], s délkou 100

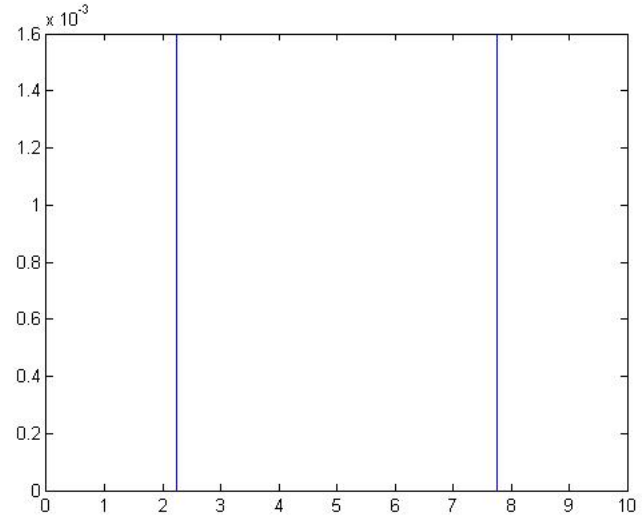


Obrázek 24: Vektor λ k úloze $F(x) = 0$, tr=1, K[1000,1000], s délkou 100

2. Nyní budu působit na strunu silou o hustotě 100, délku struny zvolím 10 a řád matice tuhosti K bude 2500. Při této možnosti se došlo k výsledku po 429 iteracích. Struna se prohla až k překážce. Takže vektor prohnutí u má nejvyšší hodnotu 250.1, ale vektor napětí λ má nejvyšší hodnotu pouze 0.0016. Pro tuto možnost neuvádím výsledky od funkce fmincon, protože se nedospělo k řešení v rozumném čase.



Obrázek 25: Vektor u k úloze $F(x) = 0$, tr=100, K[2500,2500], s délkou 10



Obrázek 26: Vektor λ k úloze $F(x) = 0$, tr=100, K[2500,2500], s délkou 10

Na závěr jsem opět vše shrnul do tabulky. Do následující tabulky jsem zapsal hodnoty, při počítání se semihladkou Newtonovou metodou. V prvním sloupci je napsán řád matice K, v druhém sloupci hustota síly, ve třetím slupci je délka struny, ve 4. sloupci je největší složka vektoru prohnutí u, v 5. sloupci je největší složka vektoru napětí λ a v posledním sloupci počet iterací za jaký se dospělo k výsledku.

řád matice tuhosti K	hustota síly	délka struny	složka s největší hodnotou vektoru u	složka s největší hodnotou vektoru λ	počet iterací
5	1	10	8	0	2
5	1	50	200	0	3
5	1	100	250.1	400	3
100	1	10	12.25	0	2
100	1	50	250.1	0.25	19
100	1	100	250.1	1	19
1000	1	10	12.45	0	2
1000	1	50	250.1	0.0025	172
1000	1	100	250.1	0.01	174
2500	100	10	250.1	0.0016	429

Následující tabulka má stejnou strukturu jako předchozí, ale s tou vyjímkou, že v ní jsou hodnoty z počítání pomocí funkce fmincon.

řád matice tuhosti K	hustota síly	délka struny	složka s největší hodnotou vektoru u	složka s největší hodnotou vektoru λ	počet iterací
5	1	10	12.5	0	7
5	1	50	250.1	62.4	5
5	1	100	250.1	625	3
100	1	10	12.4985	0	56
100	1	50	250.1	0.2551	53
100	1	100	250.1	1.0203	33
1000	1	10	12.4807	0	507
1000	1	50	250.1	0.0025	459
1000	1	100	250.1	0.01	232

Podle hodnot v tabulkách můžeme soudit, že počítání zadané praktické úlohy pomocí semihladké Newtonovy úlohy je rychlejší, než počítání pomocí minimalizační úlohy.

6 Závěr

V této práci jsem si zadefinoval potřebné definice z Clarkeova kalukulu. To byla definice zobecněné derivace, zobecněného gradientu a zobecněného Jacobiánu. Poté jsem si popsal "základní"Newtonovu metodu a semihladkou Newtonovou metodu. Definoval jsem, co je to semihladkost, superlinearita, a napsal jsem, kdy bude semihladká Newtonova metoda konvergovat. Poté jsem popsal implementaci semihladké Newtonovy metody v programu Matlab. Nakonec jsem provedl několik numerických testů. Nejprve jsem srovnával rychlost výpočtu podle iterací, při použití analyticky spočteného Jacobiánu a při využití numericky spočteného Jacobiánu. Výpočet s analyticky spočteným Jacobiánem konvergoval rychleji. Dále jsem řešil praktickou úlohu, deformaci struny při kontaktu s překážkou. Popsal jsem si tuto úlohu několika způsoby. V prvním případě se dá dospět k výsledku úlohy pomocí řešení soustavy s maximovou funkcí a u tohoto způsobu jsem používal výpočet pomocí semihladké Newtonovy metody. Další možností je řešit minimalizační úlohu. Protože má Matlab předdefinovanou funkci pro výpočet minimalizační úlohy, tak jsem tuto možnost používal jako kontrolu správnosti. Zjistil jsem, že při zvolení vhodného řádu matice tuhosti K je použití semihladké Newtonovy metody rychlejší, ve srovnání s funkcí fmincon. Při výšším řádu matice tuhosti se dospěje oběma způsoby k hodně podobným výsledkům, ale počítání pomocí semihladké Newtonovy metody je rychlejší.

Literatura

[1] Michael Hintermüller: *Semismooth Newton Methods and Applications*,Oberwolfarh, 2010.

[2] Jiří Outrata, Michal Kočvara a Jochem Zowe: *Nonsmooth Approach to Optimization Problems with Equilibrium Constraints*, Kluwer Academic Publishers, 1998.

[3] Radek Kučera, Kristýna Motyčková: *Semi-smooth Newton method for solving 2D contact problems with Tresca and Coulomb friction*, Advances in electrical and electronic engineering, 11/3, 218-226, 2013.

[4] Vít Vondrák, Lukáš Pospíšil: *Numerické metody I*, text vzniklý při realizaci projektu Matematika pro inženýry 21. století, 2011.

[5] Tomáš Kozubek, Tomáš Brzobohatý, Marta Jarošová, Václav Hapla, Alexandros Makropoulus: *Lineární algebra s Matlabem*, text vzniklý při realizaci projektu Matematika pro inženýry 21. století, 2011.